Chapter 9: Programmer's Guide to the Process Logger

This chapter describes the Process Logger and provides some guidance for programmers wishing to configure their external programs to create **CRL** entries.

9.1 Introduction

The Process Logger (Plog) provides a way to create and store **CRL** entries from programs external to **CRL**. This is particularly useful for programs that monitor alarms or devices on the experiment. Plog entries are stored and viewed in the same way as entries inserted from within the **CRL** application.

Plog is run as a standalone daemon process that monitors specified TCP ports for input, interprets the input as **CRL** entries, and creates and logs the entries. Information on starting the Plog daemon is provided in section 11.9 *Starting the Process Logger Daemon*.

There can be multiple back-to-back messages on a single open TCP connection and many concurrent TCP connections on any TCP port. Each experiment must assign and make known the TCP port number(s) for remote program connections.

9.2 Guidelines for Programmers

The Process Logger communication is full duplex. Your program needs to send messages (the entries) to Plog and to read return messages from it. You must write your **CRL** entries to one of the TCP ports (sockets) on the Plog host, as assigned by your **CRL** administrator.

Input for an entry must be furnished in the form of an XML message that identifies the entry's various header elements and body, as shown in section 9.2.1 *Entry Message Format*. The header elements include operator name, category, topic, and keyword(s), all of which are optional but recommended. These element types are described in Chapter 1: *Overview*. Currently, there is no validation of the values of these elements against values already defined for a particular **CRL** installation, so you can choose unique text strings for these items, or not. The search facilities in **CRL** will allow users to search on the operator, category, topic, and keyword(s) for your program's entries, regardless. Make these names descriptive!

☞ **Note that if the header element values you choose don't match values already defined, then you need to communicate to your fellow experimenters exactly what values to search on in order to retrieve these entries!**

Currently, only text or plaintext messages may be included in Plog entries; no binary data is accepted[1]. Text and plaintext are compared and contrasted in section 3.3.1 *Text*.

9.2.1 Entry Message Format

The format of the messages Plog receives from your program must be as shown below.

Notes:

- Use upper case for the element tags.[2]
- The `MESSAGE TYPE` (first line) must be set to `TEXT` or `PLAINTEXT`.
- The element tags `<MESSAGE>... </MESSAGE>` and `<TEXT>...</TEXT>` are required; all other element tags are optional.
- The logged text within `<TEXT>...</TEXT>` must either be contained within a `<![CDATA[...]]>` construction or it must conform to valid XML standards (e.g., `<P>` must be used as `<P/>`, `<BR>` must be `<BR/>`, and so on, and all tags with attributes must have the attribute value enclosed in double quotes, e.g., `<FONT size="10">...</FONT>`).
- The logged text within `<TEXT><![CDATA[...]]></TEXT>` can contain[3]:
    - · newlines
    - · carriage returns
    - · HTML tags (If the message type is plaintext, your browser should treat HTML tags properly, but **CRL** will treat the tags as text.)

1. In the future, binary data could be added by using base64 encoding and creating an XML tag for the encoded data.
2. <MESSAGE> and </MESSAGE> are the only element tags that are required to be upper case.
3. The CDATA construction is not strictly necessary, however it ensures that the text will be interpreted as a character string and will not be parsed.

Format

```
<MESSAGE TYPE="plaintext">
      <OPERATOR>Name of program or responsible person
      </OPERATOR>
      <CATEGORY>Category/subcategory/sub-subcategory/...
      </CATEGORY>
      <TOPIC>Topic
      </TOPIC>
      <KEYWORD>Keyword1
      </KEYWORD>
      <KEYWORD>Keyword2
      </KEYWORD>
      <TEXT><![CDATA[The logged text goes here.]]>
      </TEXT>
</MESSAGE>
```

9.2.2 Return Messages from Plog

Every time Plog receives an entry, it returns a message to the sending program. The return message is one of the following three:

`<SUCCESS/>`	successful entry
`<FAIL/>`	entry not saved, but no syntax error detected; may succeed if tried in future (e.g., occurs if database application is not currently running or filesystem not available)
`<ERROR/>`	message had a syntax error and will never result in a saved entry

9.2.3 Sample Java Program Excerpt

You should configure your program to run input and output threads, as illustrated in this annotated **Java** test program excerpt (text enclosed in brackets, e.g., `<text>`, indicates replacement by context-sensitive data):

```
// Start the program:
public static void main (String[] args) {
  try {

// Define target host and TCP port (socket):
     Socket s = new Socket ("<IP_address_of_Plog_host>",<TCP_port_number>);

// Open "write" target file (set to PrintWriter here):
     PrintWriter pw = new PrintWriter (s.getOutputStream(), true);

// Start thread to read return messages from Plog, BEFORE starting write thread:
     InputStreamReader isr = new InputStreamReader ( s.getInputStream () );
```

```
// Start thread to write messages to PrintWriter; this test program accepts
// input from keyboard, writes to pw, then flushes pw. Test program does
// not enforce entry message format -- yours must!
     new ClientTester ().start();
// Thread to send to Plog
     while ( true ) {
        char c = (char)System.in.read();
        pw.write(c);
        pw.flush();
     }

// Thread to read return messages from Plog
public void run () {
   char[] cbuffer = new char[20]
   while ( true ) {
      try {
         System.out.println("ISSUE READ" );
         int count = isr.read(cbuffer);

         System.out.println("READ: "+new String (cbuffer,0,count));
      }  catch ( Exception e) {}
   }
...
}
```